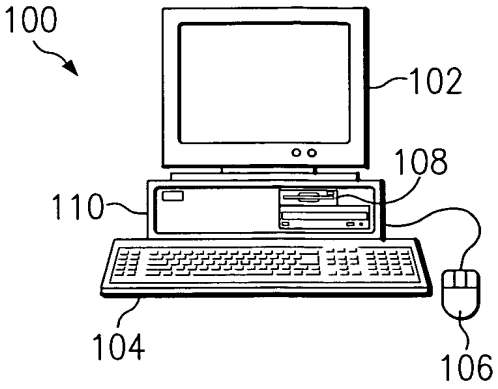
100
102
108
110
104
106

FIG. 1

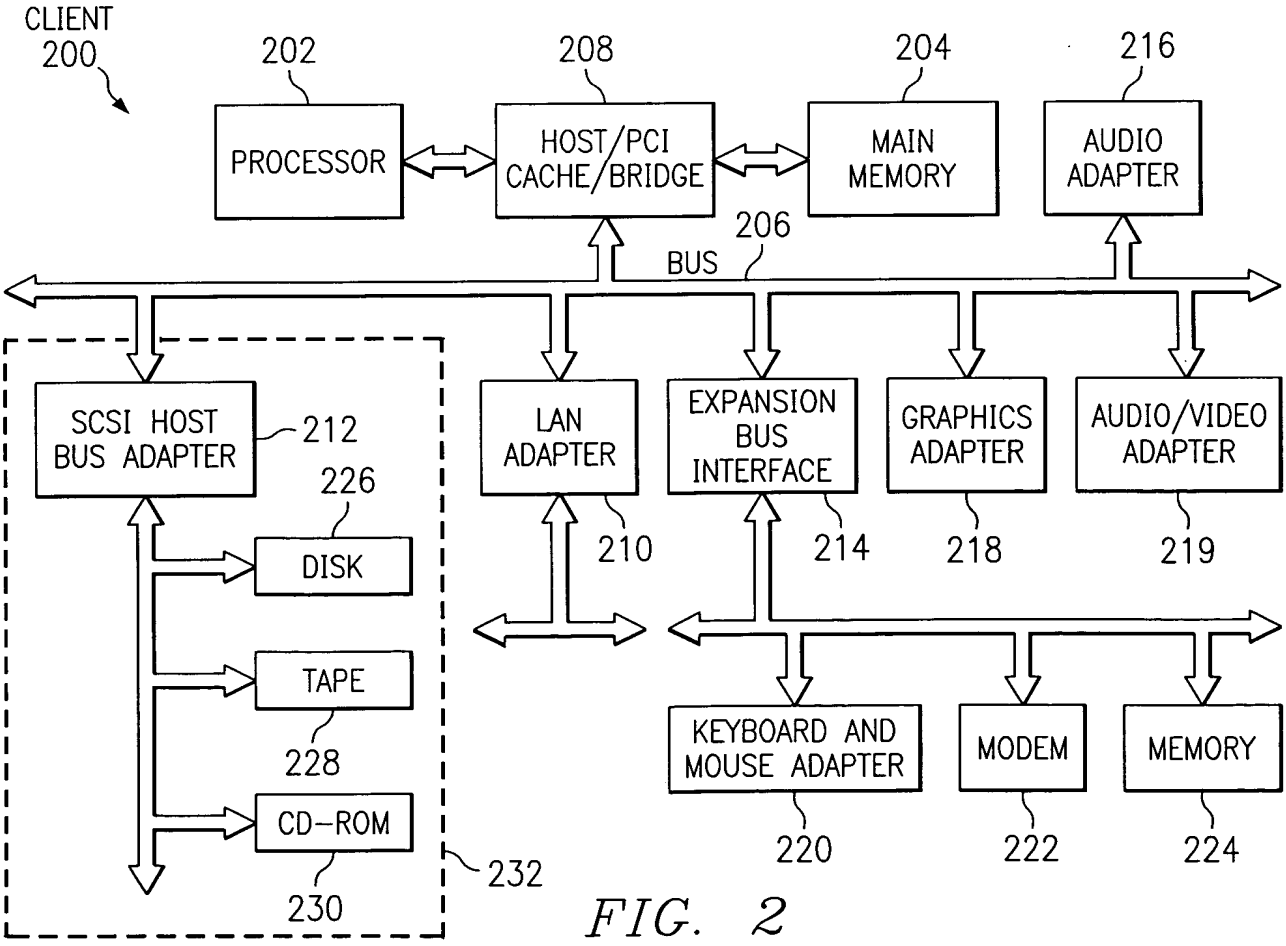
CLIENT
200
202
208
204
216
PROCESSOR
HOST/PCI
CACHE/BRIDGE
MAIN
MEMORY
AUDIO
ADAPTER
206
BUS
SCSI HOST
BUS ADAPTER
212
226
DISK
TAPE
228
CD-ROM
230
232
LAN
ADAPTER
210
EXPANSION
BUS
INTERFACE
214
GRAPHICS
ADAPTER
218
AUDIO/VIDEO
ADAPTER
219
KEYBOARD AND
MOUSE ADAPTER
220
MODEM
222
MEMORY
224

FIG. 2

FIG. 3

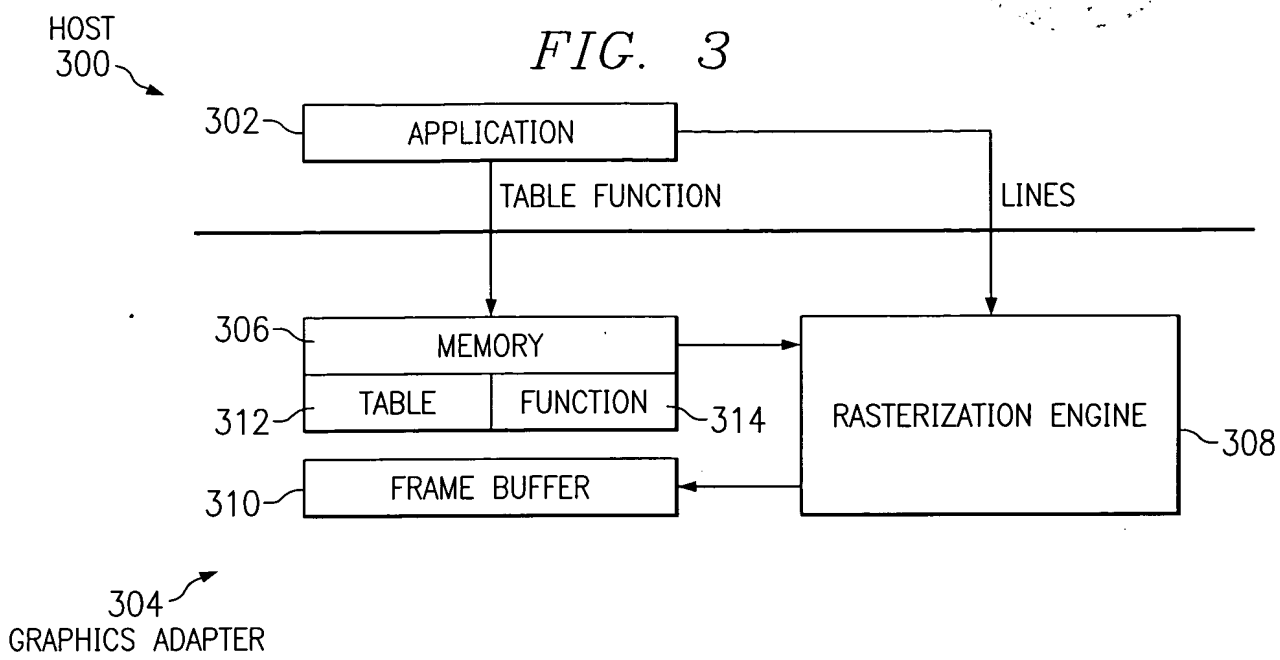

FIG. 4

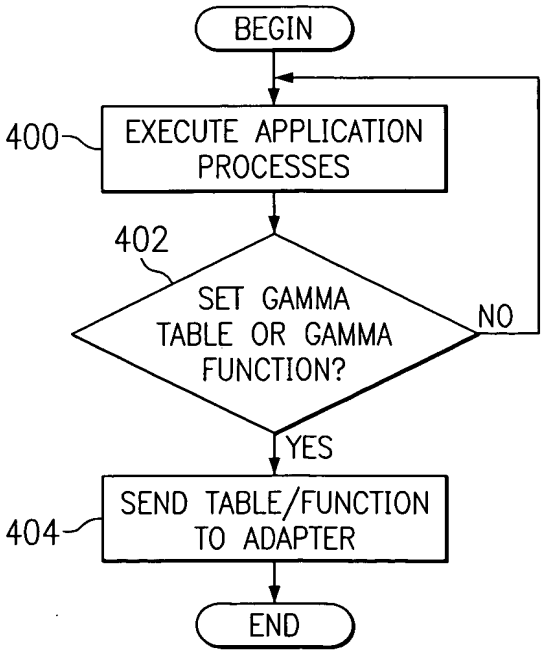

FIG. 5

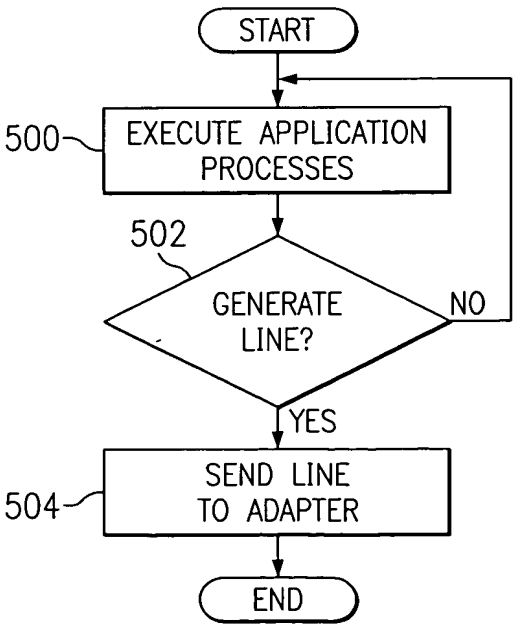

FIG. 6

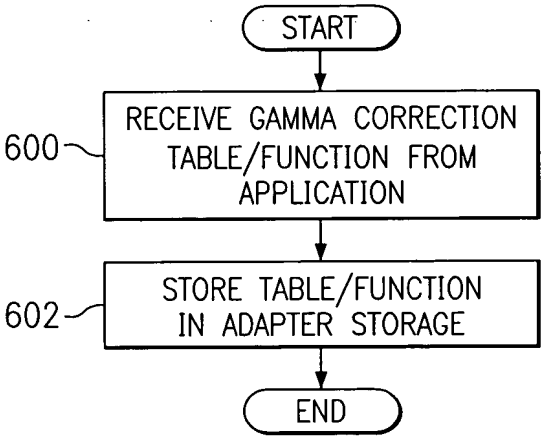

FIG. 7

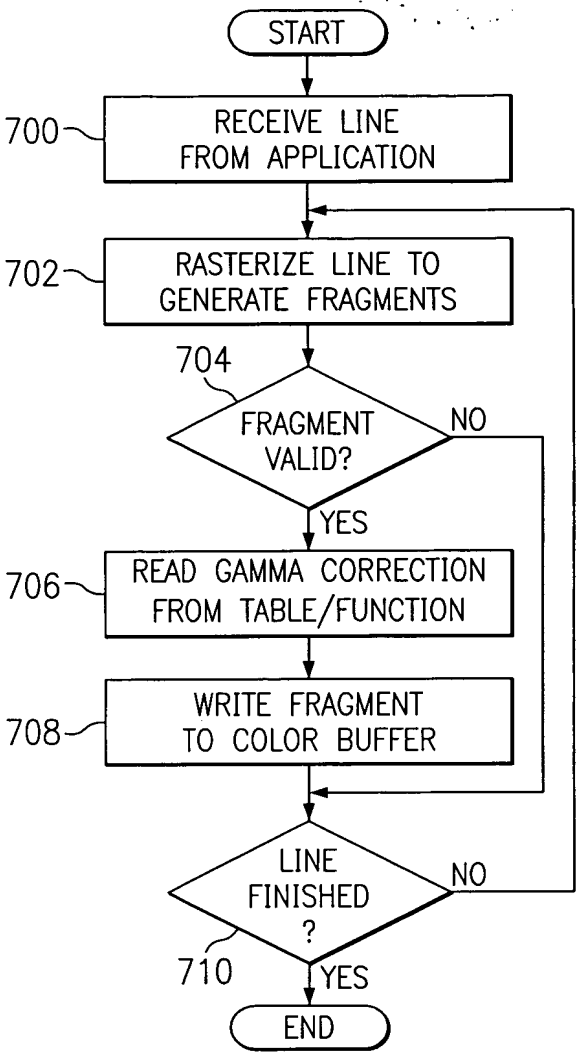

FIG. 9

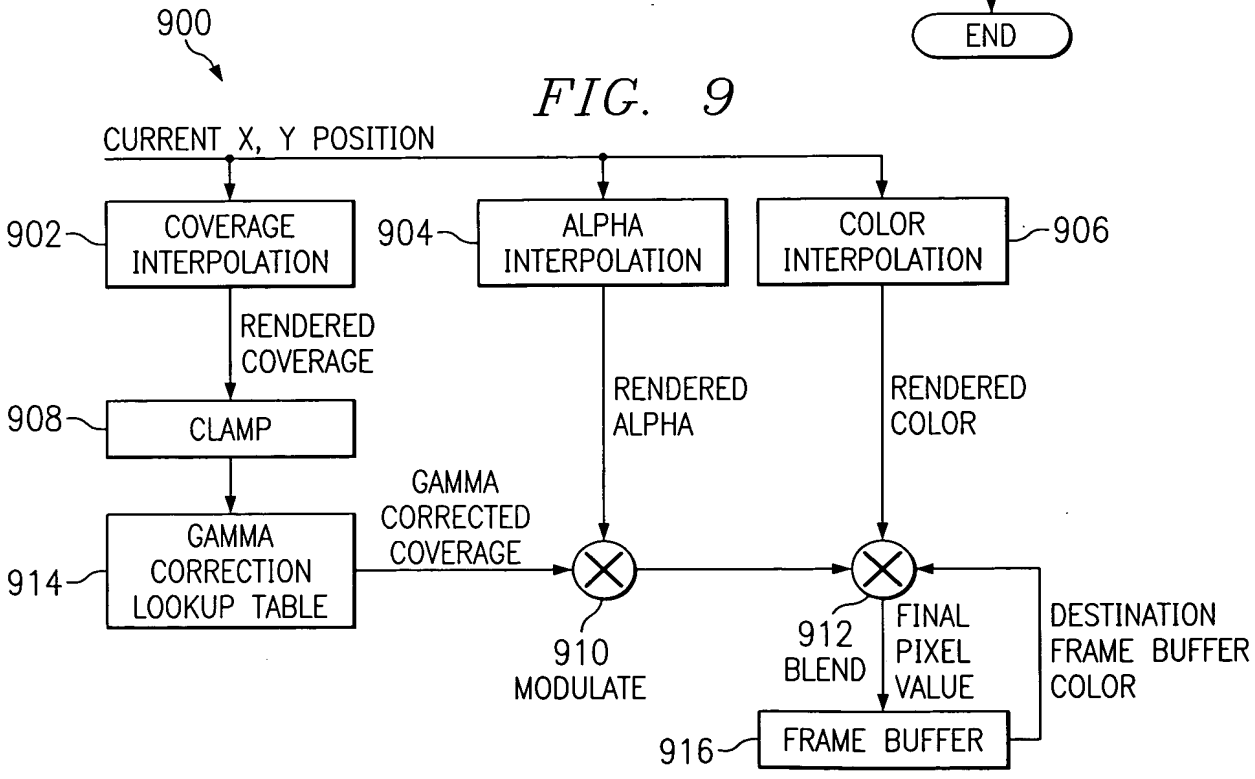

FIG. 8

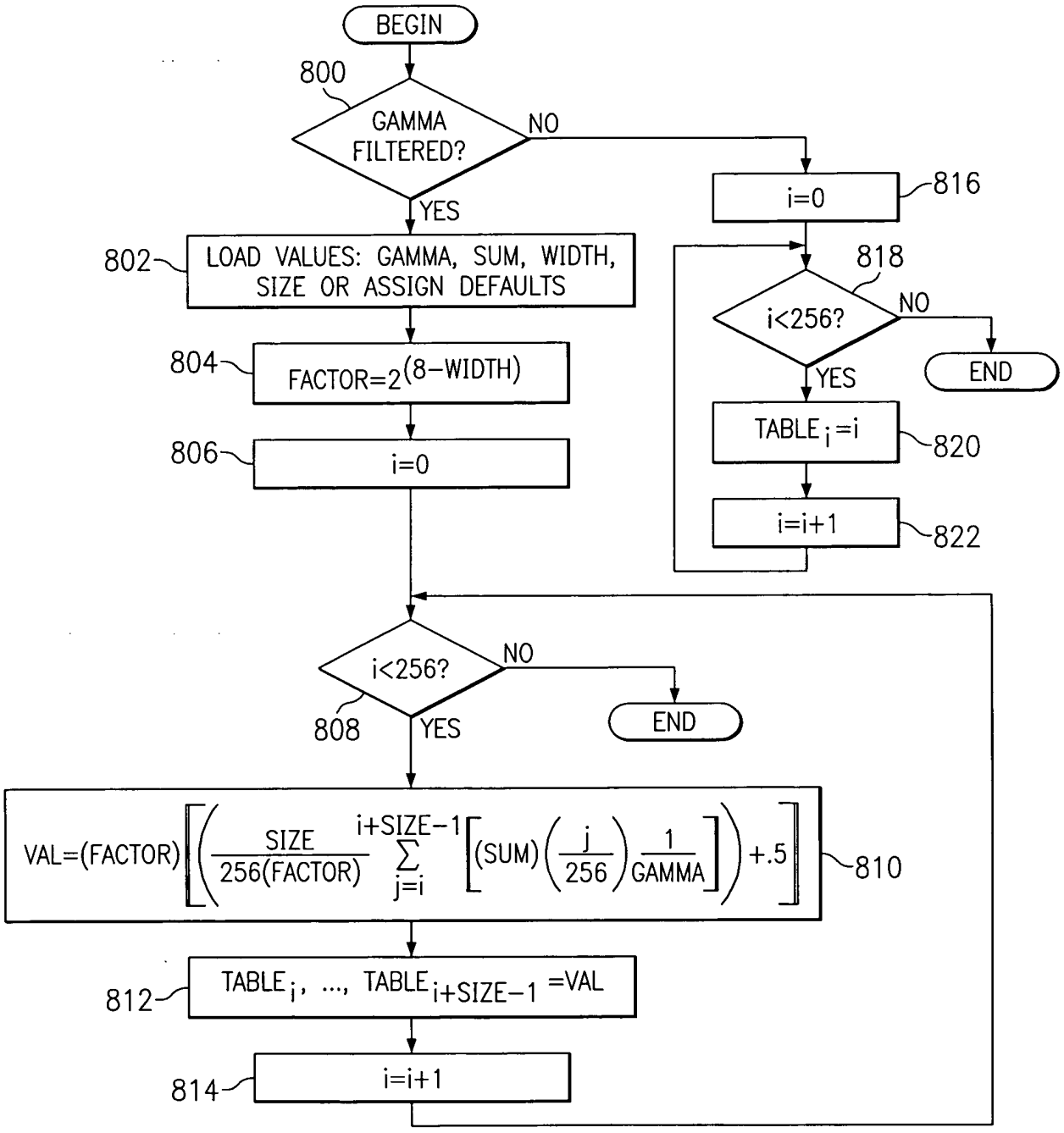
BEGIN
800
GAMMA FILTERED?
NO
YES
802
LOAD VALUES: GAMMA, SUM, WIDTH, SIZE OR ASSIGN DEFAULTS
804
FACTOR=2^(8−WIDTH)
806
i=0
808
i<256?
NO
YES
END
VAL=(FACTOR)[(SIZE/256(FACTOR)) Σ_{j=i}^{i+SIZE−1} [(SUM)(j/256)^(1/GAMMA)] +.5]
810
812
TABLE_i, ..., TABLE_{i+SIZE−1} =VAL
814
i=i+1
816
i=0
818
i<256?
NO
YES
END
820
TABLE_i=i
822
i=i+1

FIG. 10A

1000

```
if (env=getenv("_OGL_GAMMA_FILTER")){
/* Gamma filtered*/
    float gamma;
    float sum;
    int factor;
    int width;
    int size;

    gamma=1.0;
    gamma=atof(env);

    width=8;
    if (env=getenv("_OGL_GAMMA_TABLEWIDTH"))
      width=atoi(env);
    factor=(int)pow(2.0, (double) (8.0-width));

    sum=256.0;
    if (env=getenv("_OGL_GAMMA_SUM"))
      sum=atof(env);

    size=256;
    if (env=getenv("_OGL_GAMMA_TABLESIZE")){
      size=atoi(env);
     switch (size) {
     case 16:
      for (i=0; i<256; i+=16) {
        a=sum * pow((double)(i/256.0), (double)(1.0/gamma));
        b=sum * pow((double)((i+1)/256.0), (double)(1.0/gamma));
        c=sum * pow((double)((i+2)/256.0), (double)(1.0/gamma));
        d=sum * pow((double)((i+3)/256.0), (double)(1.0/gamma));
        e=sum * pow((double)((i+4)/256.0), (double)(1.0/gamma));
        f=sum * pow((double)((i+5)/256.0), (double)(1.0/gamma));
        g=sum * pow((double)((i+6)/256.0), (double)(1.0/gamma));
        h=sum * pow((double)((i+7)/256.0), (double)(1.0/gamma));
        i=sum * pow((double)((i+8)/256.0), (double)(1.0/gamma));
        j=sum * pow((double)((i+9)/256.0), (double)(1.0/gamma));
        k=sum * pow((double)((i+10)/256.0), (double)(1.0/gamma));
        l=sum * pow((double)((i+11)/256.0), (double)(1.0/gamma));
        m=sum * pow((double)((i+12)/256.0), (double)(1.0/gamma));
        n=sum * pow((double)((i+13)/256.0), (double)(1.0/gamma));
        o=sum * pow((double)((i+14)/256.0), (double)(1.0/gamma));
        p=sum * pow((double)((i+15)/256.0), (double)(1.0/gamma));
        AAFilterTable[i]=AAFilterTable[i+1]=
        AAFilterTable[i+2]=AAFilterTable[i+3]=
        AAFilterTable[i+4]=AAFilterTable[i+5]=
        AAFilterTable[i+6]=AAFilterTable[i+7]=
```

FIG. 10B

```
 AAFilterTable[i+8]=AAFilterTable[i+9]=
 AAFilterTable[i+10]=AAFilterTable[i+11]=
 AAFilterTable[i+12]=AAFilterTable[i+13]=
 AAFilterTable[i+14]=AAFilterTable[i+15]=
       (int)((((a+b+c+d+e+f+
       (int)((((a+b+c+d+e+f+
           g+h+ii+j+k+m+
           n+o+p)/(16.0*factor))+0.5)*factor);

}
break;
case 32;
 for (i=0; i<256; i+=8) {
  a=sum * pow((double)(i/256.0), (double)(1.0/gamma));
  b=sum * pow((double)((i+1)/256.0), (double)(1.0/gamma));
  c=sum * pow((double)((i+2)/256.0), (double)(1.0/gamma));
  d=sum * pow((double)((i+3)/256.0), (double)(1.0/gamma));
  e=sum * pow((double)((i+4)/256.0), (double)(1.0/gamma));
  f=sum * pow((double)((i+5)/256.0), (double)(1.0/gamma));
  g=sum * pow((double)((i+6)/256.0), (double)(1.0/gamma));
  h=sum * pow((double)((i+7)/256.0), (double)(1.0/gamma));
 AAFilterTable[i]=AAFilterTable[i+1]=AAFilterTable[i+2]=
 AAFilterTable[i+3]=AAFilterTable[i+4]=AAFilterTable[i+5]=
 AAFilterTable[i+6]=AAFilterTable[i+7]=(int)((((a+b+c+d+e+f+g+h)/(8.0*factor))+0.5)*factor);
}
break;
case 64;
 for (i=0; i<256; i+=4) {
  a=sum * pow((double)(i/256.0), (double)(1.0/gamma));
  b=sum * pow((double)((i+1)/256.0), (double)(1.0/gamma));
  c=sum * pow((double)((i+2)/256.0), (double)(1.0/gamma));
  d=sum * pow((double)((i+3)/256.0), (double)(1.0/gamma));
 AAFilterTable[i]=AAFilterTable[i+1]=
 AAFilterTable[i+2]=AAFilterTable[i+3]=
   (int)((((a+b+c+d)/4.0*factor))+0.5*factor);
}
break;
case 128;
 for (i=0; i<256; i+=2) {
  a=sum * pow((double)(i/256.0), (double)(1.0/gamma));
  b=sum * pow((double)((i+1)/256.0), (double)(1.0/gamma));
 AAFilterTable[i]=AAFilterTable[i+1]=
   (int)((((a+b)/2.0*factor))+0.5*factor);
}
break;
case 256;
 for (i=0; i<256; i++) {
 AAFilterTable[i]=
 (int)((((sum * pow((double)(i/256.0), (double)(1.0/gamma)))/factor)+0.5)*factor);
}
break;
}
}
```

FIG. 11

1100

Assumptions: Floating point coverages are defined in the normalized 0.0 to 1.0 range in which 0.0 corresponds to no coverage and 1.0 corresponds to full coverage. Fixed point coverages are defined in the range 0 to size −1.

```
float * GenFloatingPtGammaTable(int   size,
                                float gamma)
{
 int i;
 float * table;

 if (table=malloc(sizeof(float)*size)) {
  for (i=0; i<size; i++) {
   table[i]=(float)pow((double)i/(size-1), (double)(1.0/gamma));
  }
 }
 return (table);
}

int * GenFixedPtGammaTable(int   size,
                           float gamma)
{
 int i;
 int *table;
 float val;

 if (table=malloc(sizeof(int)*size)) {
  for(i=0; i<size; i++) {
   val=(float)pow((double)i/(size-1), (double)(1.0/gamma));
     table[i]=(int)((size-1)*val+0.5);
  }
 }
 return (table);
}
```